Lab #8: Unsupervised Learning

Biostat 216: Machine Learning in R for the Biomedical Sciences

Principal Components Analysis

1. Read in the breast cancer imaging data "ispy1doctored2.csv" into a data frame called `dat`

```
dat <- read.csv("ispy1doctored2.csv")
```

2. Perform principal components on just the variables `"pixelVolT0"`, `"pixelVolT1"`, `"pixelVolT2"`, `"pixelVolTfinal"`, `"pixelVolPctChgT0_T1"`, `"ftvPeT0"`, `"ftvPeT1"`, `"ftvPeT2"`, `"ftvPeTfinal"`, `"ftvPePctChgT0_T1"`, `"age"`, `"MRI_LD_T0"`, `"MRI_LD_T1"`, `"MRI_LD_T2"`, `"MRI_LD_Tfinal"`. Use the command `pr.out <- prcomp(~ pixelVolT0 + pixelVolT1 + pixelVolT2 + pixelVolTfinal + pixelVolPctChgT0_T1 + ftvPeT0 + ftvPeT1 + ftvPeT2 + ftvPeTfinal + ftvPePctChgT0_T1 + age + MRI_LD_T0 + MRI_LD_T1 + MRI_LD_T2 + MRI_LD_Tfinal, data = dat, scale=TRUE)`. The `scale=TRUE` argument makes all the variables be normalized before performing PCA.

```
pr.out <- prcomp(~ pixelVolT0 + pixelVolT1 + pixelVolT2 + pixelVolTfinal + pixelVolPctChgT0_T1 + ftvPeT
```

3. Look at the `center` and `scale` components of `pr.out`. These correspond to the means and standard deviations of the variables that were used for scaling prior to implementing PCA. E.g. `pr.out$center`

```
pr.out$center
```

```
##          pixelVolT0          pixelVolT1          pixelVolT2
##         0.001337278         0.001271772         0.001307040
##      pixelVolTfinal pixelVolPctChgT0_T1             ftvPeT0
##         0.001319201        -2.597529980        25.828501078
##             ftvPeT1             ftvPeT2          ftvPeTfinal
##        15.819718095         6.416387467         2.926144333
##    ftvPePctChgT0_T1                 age           MRI_LD_T0
##       -33.060239968        48.227769231        65.800000000
##           MRI_LD_T1           MRI_LD_T2       MRI_LD_Tfinal
##        58.230769231        43.792307692        29.569230769
```

```
pr.out$scale
```

```
##          pixelVolT0          pixelVolT1          pixelVolT2
##        5.455696e-04        5.046348e-04        5.075021e-04
##      pixelVolTfinal pixelVolPctChgT0_T1             ftvPeT0
##        5.148057e-04        2.157583e+01        3.907992e+01
##             ftvPeT1             ftvPeT2          ftvPeTfinal
##        2.158310e+01        2.028899e+01        1.025553e+01
##    ftvPePctChgT0_T1                 age           MRI_LD_T0
##        4.253753e+01        9.047455e+00        2.775290e+01
##           MRI_LD_T1           MRI_LD_T2       MRI_LD_Tfinal
##        2.792391e+01        2.975864e+01        2.912708e+01
```

4. What is the dimension of the part of the data frame that you performed PCA on? Hint: you can find it as an object in `pr.out` – check `?pr.out`

```
dim(pr.out$x)
```

```
## [1] 130  15
```

5. Now look at the `rotation` matrix component of `pr.out` which gives the principal component loading vector. How many PCs are there? Does that agree with what you would expect?

```
pr.out$rotation
```

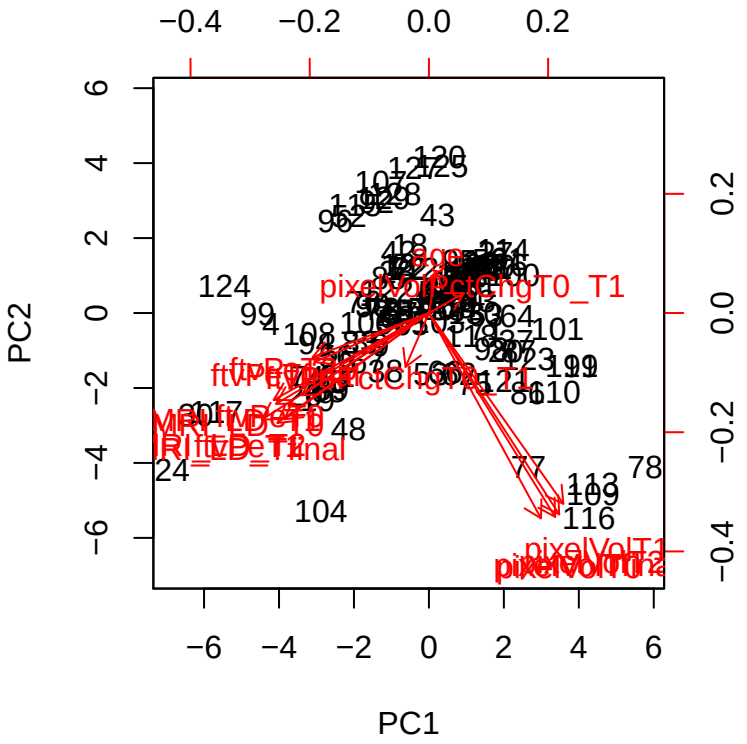
```
##                             PC1         PC2          PC3         PC4
## pixelVolT0           0.23499941 -0.43058919  0.045760336 -0.14896734
## pixelVolT1           0.28198315 -0.40013543 -0.052002439  0.19136918
## pixelVolT2           0.27334862 -0.42229813 -0.002142282  0.04233403
## pixelVolTfinal       0.26485372 -0.42744674 -0.005907989  0.03295573
## pixelVolPctChgT0_T1  0.07320258  0.04158706 -0.218116822  0.60668093
## ftvPeT0             -0.26646345 -0.17242729  0.218057337  0.25188759
## ftvPeT1             -0.30287387 -0.22278701  0.045780306  0.18167754
## ftvPeT2             -0.24385157 -0.09257910 -0.531657645  0.04315225
## ftvPeTfinal         -0.24027040 -0.10276296 -0.507771678  0.07572300
## ftvPePctChgT0_T1    -0.04901623 -0.11300048 -0.414476342 -0.26404896
## age                  0.01519258  0.08783855  0.055429426  0.62224518
## MRI_LD_T0           -0.32046697 -0.19049676  0.319947454  0.01088290
## MRI_LD_T1           -0.32552700 -0.18227543  0.273912354 -0.01011548
## MRI_LD_T2           -0.35049209 -0.22373157  0.024623024 -0.06749089
## MRI_LD_Tfinal       -0.31774420 -0.23056117 -0.067726102 -0.06715415
##                             PC5          PC6          PC7          PC8
## pixelVolT0           0.13333420 -0.205878033  0.095815606 -0.017415116
## pixelVolT1          -0.04855158  0.023353387 -0.103890024 -0.022036833
## pixelVolT2          -0.05681500  0.004421023  0.004061152  0.051840037
## pixelVolTfinal      -0.06954453 -0.031143248  0.005327724  0.016175839
## pixelVolPctChgT0_T1 -0.31740435  0.457326468 -0.364420382 -0.006103738
## ftvPeT0              0.55102127  0.074098054 -0.172649400 -0.096552740
## ftvPeT1              0.50483266  0.221559799 -0.003627574  0.005387928
## ftvPeT2              0.02823859 -0.321114767 -0.136001995  0.132665584
## ftvPeTfinal          0.01925239 -0.399588694 -0.131853147  0.051478587
## ftvPePctChgT0_T1     0.10602171  0.579608468  0.418995934  0.405485046
## age                 -0.02386982 -0.255144104  0.724291153  0.098623574
## MRI_LD_T0           -0.23670210 -0.067844549 -0.107487785  0.422864048
## MRI_LD_T1           -0.32394111 -0.026782648 -0.044171347  0.417975333
## MRI_LD_T2           -0.24609183  0.087458127  0.150728556 -0.359781379
## MRI_LD_Tfinal       -0.27940833  0.124712539  0.212361189 -0.556343752
##                              PC9          PC10         PC11         PC12
## pixelVolT0           1.664792e-01 -0.121597040  0.426760919 -0.09019039
## pixelVolT1           1.552435e-01 -0.181290986  0.450335630  0.04256503
## pixelVolT2          -7.905710e-02  0.313728118 -0.349182242 -0.35977763
## pixelVolTfinal      -2.175290e-01  0.017611007 -0.508577700  0.40798663
## pixelVolPctChgT0_T1  5.508080e-02 -0.030269564  0.054141830 -0.03876338
## ftvPeT0              1.540007e-06 -0.054898518 -0.195626441 -0.18048980
## ftvPeT1             -9.765186e-02  0.126718224  0.158326214  0.17384250
## ftvPeT2              2.981823e-02  0.433946990  0.126887653  0.44023546
## ftvPeTfinal         -4.167520e-02 -0.408321120 -0.179917734 -0.45155447
## ftvPePctChgT0_T1     4.523253e-02 -0.124535425 -0.032491311 -0.06582404
## age                  1.752564e-02  0.003023469 -0.003861423  0.01443236
## MRI_LD_T0            1.636641e-01 -0.483563713 -0.129167213  0.32826126
## MRI_LD_T1           -2.827817e-01  0.381002176  0.236316325 -0.33058049
## MRI_LD_T2            7.007297e-01  0.215849786 -0.168147603 -0.06885642
## MRI_LD_Tfinal       -5.284816e-01 -0.193604399  0.133718126  0.07635822
##                             PC13        PC14         PC15
## pixelVolT0           0.007221046 -0.00641764 -0.657709341
```

```
## pixelVolT1           -0.123735312 -0.12797758  0.643117183
## pixelVolT2            0.609787747  0.03058230  0.101866461
## pixelVolTfinal       -0.502091465  0.08572695 -0.088011954
## pixelVolPctChgT0_T1   0.011989560  0.02363805 -0.354197852
## ftvPeT0              -0.109392245 -0.59338554 -0.045730839
## ftvPeT1               0.087023096  0.65406752  0.057312446
## ftvPeT2               0.168096480 -0.28060504 -0.036500316
## ftvPeTfinal          -0.153408971  0.24064462  0.043449376
## ftvPePctChgT0_T1     -0.049941644 -0.17525383  0.001378595
## age                   0.008000843 -0.03264117 -0.011426576
## MRI_LD_T0             0.348950586 -0.01985118 -0.017569720
## MRI_LD_T1            -0.335988337 -0.01468026  0.002685508
## MRI_LD_T2            -0.152922303  0.08234887  0.027778408
## MRI_LD_Tfinal         0.169198812 -0.13786736 -0.020580365
```

There are 15 PCs. This is what is expected because it matches the number of variables.

6. Plot the first two principal components with `biplot(pr.out, scale=0)`. The `scale=0` argument ensures that arrows are scaled to represent the loadings.

```
biplot(pr.out, scale=0)
```



7. The red arrows represent the loadings and there contributions to each PC. Why do you think there are many arrows that are almost overlapping?

This is likely due to colinearity (multiple variables explaining the same part of the variance – at least with respect to the first 2 PCs)

8. Look at the `sdev` component of `pr.out`. This gives the standard deviation of each PC.

```
pr.out$sdev
```

```
##  [1] 2.1646749 1.8083891 1.3658213 1.1224987 1.0699259 0.9595897 0.8894578
##  [8] 0.6686935 0.4298787 0.3505364 0.3290380 0.2838988 0.2451433 0.1931840
```

```
## [15] 0.1436968
```

9. Generate a vector `pr.var` that is the variance explained by each PC by squaring the standard deviations.

```
pr.var <- pr.out$sdev^2
pr.var
```

```
##  [1] 4.68581758 3.27027127 1.86546779 1.26000327 1.14474143 0.92081241
##  [7] 0.79113523 0.44715097 0.18479571 0.12287575 0.10826599 0.08059855
## [13] 0.06009522 0.03732005 0.02064877
```

10. Generate a new vector `pve` that is the proportion of variance explained by each vector. Generate the `sum` of `pve` as a confirmation that you are on the right lines.

```
pve <- pr.var/sum(pr.var)
pve
```

```
##  [1] 0.312387839 0.218018085 0.124364519 0.084000218 0.076316095
##  [6] 0.061387494 0.052742349 0.029810065 0.012319714 0.008191717
## [11] 0.007217733 0.005373237 0.004006348 0.002488003 0.001376585
```

11. Generate plots of proportion of variance explained and cummulative variance explained against PC number. You will need the function `cumsum` which compute the cumulative sum of elements in a numeric vector, e.g. for `a1 <- c(2,1,3,4)`, `cumsum(a1)` gives `[1]  2  3  6 10`

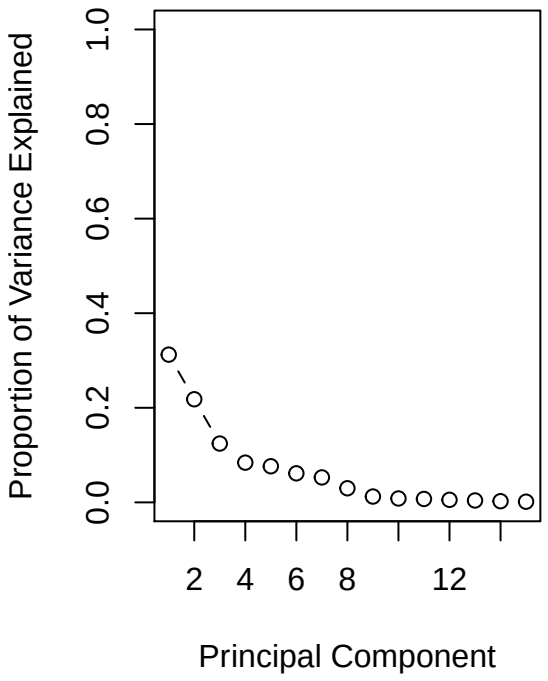
```
a1 <- c(2,1,3,4)
a1
```

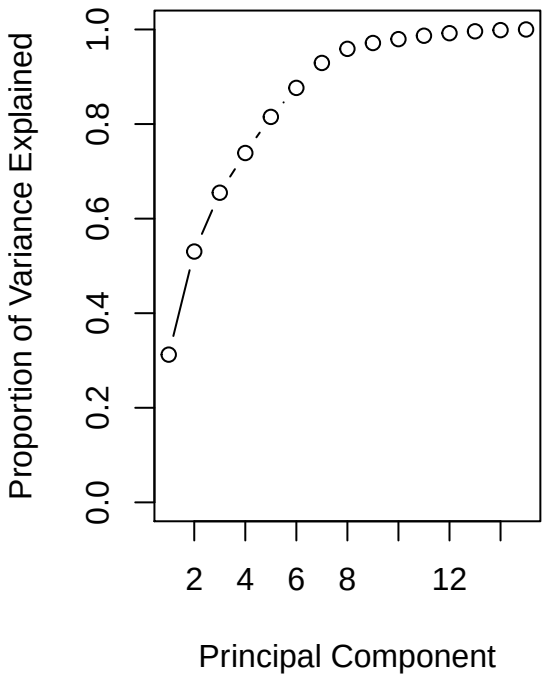
```
## [1] 2 1 3 4
```

```
cumsum(a1)
```

```
## [1]  2  3  6 10
```

```
par(mfrow=c(1,2))
plot(1:length(pve),pve, xlab="Principal Component", ylab="Proportion of Variance Explained", ylim=c(0,1)
plot(1:length(pve),cumsum(pve), xlab="Principal Component", ylab="Proportion of Variance Explained", yli
```





```
par(mfrow=c(1,1))
```

12. How many PCs do you you need to explain at least 80% of the variance?

5 PCs

K-means clustering

13. Subset `dat` to only the columns `"pixelVolTfinal"`, `"ftvPeTfinal"`, `"age"`, `"MRI_LD_Tfinal"`. Name the new object `datsub`

```
datsub <- dat[ ,c("pixelVolTfinal", "ftvPeTfinal", "age", "MRI_LD_Tfinal")]
```

14. Perform K-means clustering on the subset of variables for $K = 3$. Use 30 random starting configurations.

```
kmeans3 <- kmeans(datsub, centers=3, nstart=30)
```

15. Output the cluster assignments of the 3 clusters

```
kmeans3$cluster
```

```
##   [1] 2 2 3 3 3 2 2 3 3 2 2 3 2 3 3 3 3 2 1 2 3 3 3 1 1 1 2 3 2 1 2 3 2 2 3
##  [36] 2 2 1 3 2 2 3 2 2 2 2 2 1 3 2 2 3 2 2 3 1 3 3 2 2 2 3 2 2 3 2 3 1 2 3
##  [71] 2 2 3 3 3 3 3 2 3 2 2 2 2 2 2 3 2 2 1 3 2 2 2 1 2 3 2 2 3 2 2 3 3 1 3
## [106] 3 2 1 2 2 2 2 2 2 2 2 1 2 2 2 3 2 2 1 2 2 2 2 3 2
```

16. Generate a table of agreement between the cluster labels and the outcome pathologic complete response `pCR`. Do you think the cluster allocations are illuminating with respect to pCR?

```
table(kmeans3$cluster,dat$pCR)
```

```
##
##      0  1
##   1 14  1
##   2 44 28
##   3 36  7
```

There is some information in the clusters with respect to pCR. Patients in cluster 1 appear to have a greater tendency toward pCR than either of the other two clusters.

Hierarchical clustering

17. The `hclust()` function implements hierarchical clustering in `R`. We will plot the hierarchical clustering dendogram using complete, single, and average linkage clustering, with Euclidean distance as the similarity measure. Start by creating the distance matrix between all pairs of observations using the `dist()` function: `distx <- dist(datsub)`.

```
distx <- dist(datsub)
```

18. Generate hierarchical clustering with `method="complete"`. Use the command `hc.complete <- hclust(distx, method="complete")`.

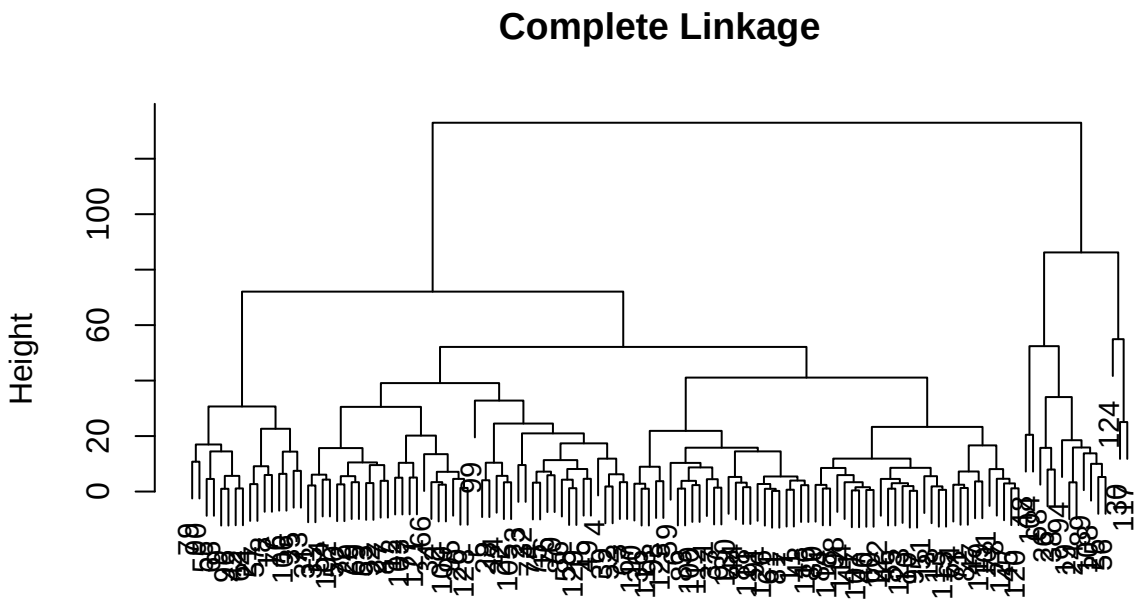
```
hc.complete <- hclust(distx, method="complete")
```

19. Generate similar objects for `average` and `single` linkage.

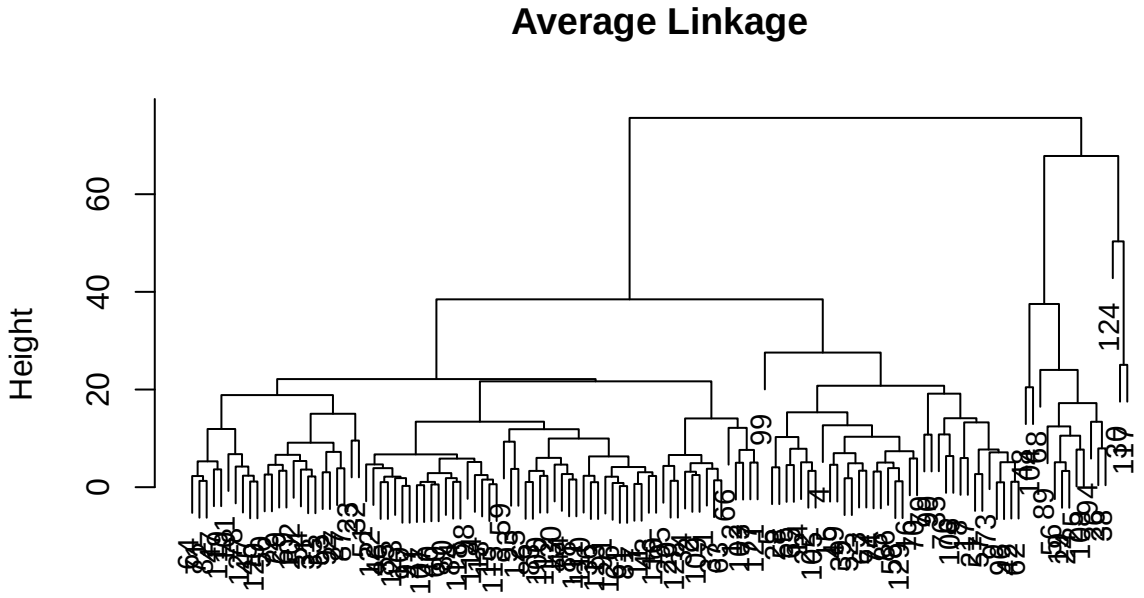
```
hc.average <- hclust(distx, method="average")
hc.single <- hclust(distx, method="single")
```

20. Plot each of the dendograms using the `plot` function.

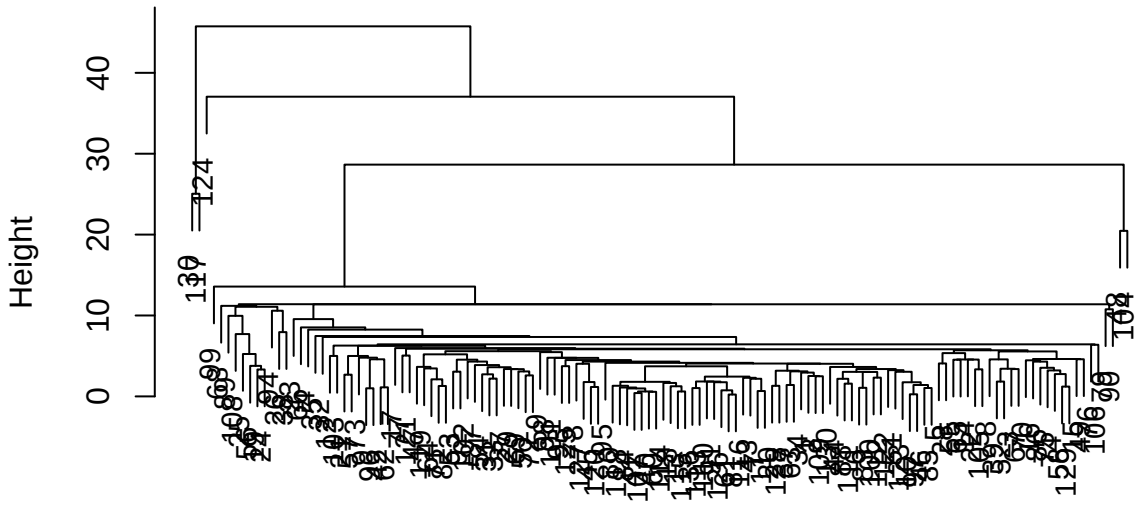
```
plot(hc.complete,main="Complete Linkage", xlab="", sub="", cex=.9)
```




```
plot(hc.average, main="Average Linkage", xlab="", sub="", cex=.9)
```




```
plot(hc.single, main="Single Linkage", xlab="", sub="", cex=.9)
```



21. Determine the cluster labels for each observation associated with cutting the dendogram with 4 clusters using the `cutree()` function.

```
cutree(hc.complete, k=4)
```

```
##   [1] 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 3 1 2 1 1 3 3 3 1 1 1 4 1 1 1 1 2
##  [36] 1 1 3 1 1 1 2 1 1 1 1 1 3 1 1 1 1 1 1 2 3 2 1 1 1 1 2 1 1 2 1 1 3 1 1
##  [71] 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 3 2 1 1 1 3 1 2 1 1 1 1 1 1 1 3 1
## [106] 2 1 3 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 4 1 1 1 1 1 1
```

```
cutree(hc.average, k=4)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 3 1 1 1 1 1
##  [36] 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1
## [106] 1 1 2 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 4 1 1 1 1 1 1
```

```
cutree(hc.single, k=4)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 4 1 1 1 1 1 1
```